In [ ]:
```
import pandas as pd
import numpy as np;
```

```
C:\Users\UMMAR\anaconda3\lib\site-packages\numpy\_distributor_init.py:30: UserWarning: loaded more than 1 DLL fr
om .libs:
C:\Users\UMMAR\anaconda3\lib\site-packages\numpy\.libs\libopenblas.EL2C6PLE4ZYW3ECEVIV3OXXGRN2NRFM2.gfortran-win
_amd64.dll
C:\Users\UMMAR\anaconda3\lib\site-packages\numpy\.libs\libopenblas.WCDJNK7YVMPZQ2ME2ZZHJJRJ3JIKNDB7.gfortran-win
_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

In [ ]:
```
df = pd.read_csv("diabetes.csv")
```

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(768, 9)
```

In [ ]:
```
for col in df.columns:
        missing_rows = df.loc[df[col]==0].shape[0]
        print(col + ": " + str(missing_rows))
```

```
Pregnancies: 111
Glucose: 5
BloodPressure: 35
SkinThickness: 227
Insulin: 374
BMI: 11
DiabetesPedigreeFunction: 0
Age: 0
Outcome: 500
```

In [ ]:
```
df['Glucose'] = df['Glucose'].replace(0, np.nan)
df['BloodPressure'] = df['BloodPressure'].replace(0, np.nan)
df['SkinThickness'] = df['SkinThickness'].replace(0, np.nan)
df['Insulin'] = df['Insulin'].replace(0, np.nan)
df['BMI'] = df['BMI'].replace(0, np.nan)

df['Glucose'] = df['Glucose'].fillna(df['Glucose'].mean())
df['BloodPressure'] = df['BloodPressure'].fillna(df['BloodPressure'].mean())
df['SkinThickness'] = df['SkinThickness'].fillna(df['SkinThickness'].mean())
df['Insulin'] = df['Insulin'].fillna(df['Insulin'].mean())
df['BMI'] = df['BMI'].fillna(df['BMI'].mean())
```

In [ ]:
```
for col in df.columns:
        missing_rows = df.loc[df[col]==0].shape[0]
        print(col + ": " + str(missing_rows))
```

```
Pregnancies: 111
Glucose: 0
BloodPressure: 0
SkinThickness: 0
Insulin: 0
BMI: 0
DiabetesPedigreeFunction: 0
Age: 0
Outcome: 500
```

In [ ]:
```
df.describe()
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	76
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.457464	0.471876	33.240885	
std	3.369578	30.435949	12.096346	8.790942	85.021108	6.875151	0.331329	11.760232	
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	
25%	1.000000	99.750000	64.000000	25.000000	121.500000	27.500000	0.243750	24.000000	
50%	3.000000	117.000000	72.202592	29.153420	155.548223	32.400000	0.372500	29.000000	
75%	6.000000	140.250000	80.000000	32.000000	155.548223	36.600000	0.626250	41.000000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	

In [ ]:
```
from sklearn import preprocessing
```

In [ ]:
```
df_scaled = preprocessing.scale(df)
df_scaled = pd.DataFrame(df_scaled, columns=df.columns)
df_scaled['Outcome'] = df['Outcome']
df = df_scaled
```

In [ ]:
```
X = df.loc[:, df.columns != 'Outcome']
y = df.loc[:, 'Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state= 13)

# Build neural network in Keras
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=8))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))

model.add(Dense(64, activation = "relu"))

model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=200, verbose=False)

# Results - Accuracy
scores = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: %.2f%%\n" % (scores[1]*100))
scores = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy: %.2f%%\n" % (scores[1]*100))
```

```
Training Accuracy: 98.86%

Testing Accuracy: 72.73%
```

In [ ]:
```
(model.predict(X_test) > 0.5).astype("int32")
```

```
5/5 [==============================] - 0s 1ms/step
```

Out[ ]:
```
array([[1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [1],
       [0],
       [1],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [0],
       [0],
       [1],
       [1],
       [0],
       [0],
       [1],
       [0],
       [1],
       [1],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [1],
       [1],
       [1],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
```

```
[1],
[1],
[0],
[0],
[1],
[0],
[0],
[0],
[1],
[0],
[0],
[0],
[1],
[1],
[0],
[1],
[0],
[1],
[0],
[0],
[1],
[1],
[0],
[1],
[0],
[0],
[0],
[0],
[0],
[1],
[1],
[1],
[1],
[0],
[1],
[0],
[1],
[0],
[0],
[0],
[1],
[0],
[0],
[1],
[0],
[1],
[0],
[1],
[1],
[0],
[0],
[0],
[0],
[1],
[1],
[1],
[0],
[1],
[0],
[1],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[1],
[0],
[0],
[0]])
```

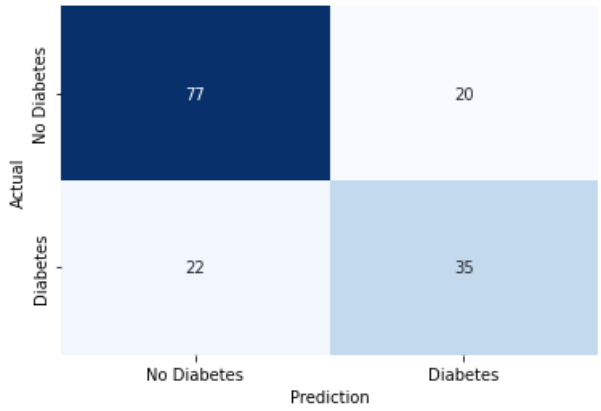
In [ ]:
```
# Results - Confusion Matrix
y_test_pred = (model.predict(X_test) > 0.5).astype("int32")
```

```
5/5 [==============================] - 0s 1ms/step
```

In [ ]:
```
```

In [ ]:
```
c_matrix = confusion_matrix(y_test, y_test_pred)
ax = sns.heatmap(c_matrix, annot=True, xticklabels=['No Diabetes', 'Diabetes'], yticklabels=['No Diabetes', 'Di
```

```
ax.set_xlabel("Prediction")
ax.set_ylabel("Actual")
plt.show()
plt.clf()
```

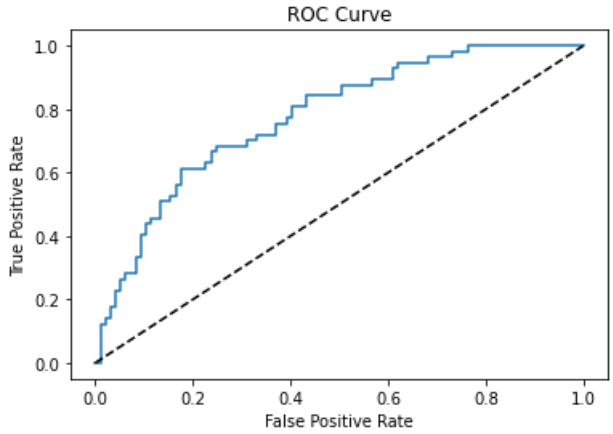



```
<Figure size 432x288 with 0 Axes>
```

In [ ]:

In [ ]:
```
# Results - ROC Curve
y_test_pred_probs = model.predict(X_test)
FPR, TPR, _ = roc_curve(y_test, y_test_pred_probs)
plt.plot(FPR, TPR)
plt.plot([0,1],[0,1],'--', color='black') #diagonal line
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
plt.clf();
```

```
5/5 [==============================] - 0s 1ms/step
```




```
<Figure size 432x288 with 0 Axes>
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: